Embedded Linux kernel解析ノウハウ
Linux kernel traceの詳細

Yoichi Yuasa

OSAKA NDS Embedded Linux Cross Forum #4

自己紹介

- 湯浅陽一
- 1999年よりLinux kernel開発に参加
- MIPSアーキテクチャのいくつかのCPUへLinux kernelを移植

Linux kernel tracerとは

- Linux kernel内蔵の内部追跡システム
- 用途に応じて複数追跡方法から選択可能
- 時間計測が可能
- システム動作中にON/OFF可能
- メモリ上のリングバッファに記録
- フィルタ機能があり選択記録が可能
- インタフェースはメモリファイルシステム上のファイル
- ARMアーキテクチャなど組込みでも利用可能
- kernelのデバッグ、検証、分析やチューニングに利用

tracerの種類

- Kernel Function Tracer
  - Kernel Function Graph Tracer
- Interrupts-off Latency Tracer
- Preemption-off Latency Tracer
- Scheduling Latency Tracer
- Trace Events(デフォルトで組込まれている)
  - Trace gpio events

Kernel configuration

Kernel hacking --->

[*] Tracers --->

    [*] Kernel Function Tracer

    [*]   Kernel Function Graph Tracer

    [ ] Interrupts-off Latency Tracer

    [ ] Preemption-off Latency Tracer

    [ ] Schedulin Latency Tracer

    [ ] Trace gpio events

Kernel Function Tracer

- trace内容
  - 実行プロセス(タスク)
  - 実行CPU番号
  - 割込み禁止/許可
  - スケジュール要求
  - ハードIRQ/ソフトIRQ
  - プリエンプト深さ
  - タイムスタンプ
  - 関数および呼び出し関係

Kernel Function Trace表示

```
# tracer: function
#
# entries-in-buffer/entries-written: 44736/44736   #P:4
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
 rcar3_create_ga-2030  [000] ...1   249.796155: mutex_unlock <-rb_simple_write
 rcar3_create_ga-2030  [000] ...1   249.796164: __fsnotify_parent <-vfs_write
 rcar3_create_ga-2030  [000] ...1   249.796165: fsnotify <-vfs_write
 rcar3_create_ga-2030  [000] ...1   249.796167: __sb_end_write <-vfs_write
 rcar3_create_ga-2030  [000] ...1   249.796167: percpu_up_read <-__sb_end_write
 rcar3_create_ga-2030  [000] ...1   249.796168: update_fast_ctr <-percpu_up_read
```

Kernel Function Graph Tracer

- trace内容
  - 実行CPU番号
  - 実行時間(処理時間)
  - 関数呼び出し階層表示
  - 実行プロセス切り替え

Kernel Function Graph Trace表示

```
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
 1)   0.720 us    |  mutex_unlock();
 1)   0.120 us    |  __fsnotify_parent();
 1)   0.120 us    |  fsnotify();
 1)               |  __sb_end_write() {
 1)               |    percpu_up_read() {
 1)               |      update_fast_ctr() {
 1)   0.120 us    |        preempt_count_add();
 1)   0.120 us    |        preempt_count_sub();
 1)   2.640 us    |      }
 1)   4.320 us    |    }
 1)   5.520 us    |  }
```

Kernel Function Graph Trace
実行プロセス切り替え表示

```
3)   1.560 us    |  }
 ------------------------------------------
 0)    <idle>-0    =>  kworker-1018
 ------------------------------------------

 0)               |  finish_task_switch() {
 3)               |  ep_scan_ready_list.isra.2() {
 0)               |    _raw_spin_unlock_irq() {
 1)   1.440 us    |                              }
 3)   0.120 us    |    mutex_lock();
 0)   0.120 us    |      preempt_count_sub();
```

Kernel Function Tracerの仕組み

- コンパイル時にプローブ関数を挿入
  - プロファイリング用コンパイルオプションを利用
  - mcount/_mcount/__mcount関数が各関数先頭に挿入
  - 本番環境とはバイナリが異なる
  - 最適化で展開された関数は対象外
- プローブ関数内にTracer機能を実装

前回のkprobeおさらい

- Linux kernelデバッグ機能の1つ
  - 複数のアーキテクチャで動作
- ブレイクポイント命令などを利用してkernelを動的に変更
- 指定位置に処理(プローブ)を追加
- Loadable moduleとして後からプローブを追加可能
- プローブ内にてkernel内データを変更可能

Tracerとkprobeとの違い

- 本番環境とはバイナリが異なる
  - kprobeはkprobe機能は追加されているがそれ以外は同じ
- 後からプローブを追加できない
  - kprobeは追加可能
- Kernel内のデータは変更できない
  - kprobeは変更可能

Tracerはどんな時に利用する?

- 初期調査
  - 特定のアプリケーション処理時にどんな処理が行われるか知りたい
    - システムコール呼び出し時
    - ファイルシステム操作時(sysfsなどのデバイスIFを含む)
  - 他レイヤからどのように処理が呼び出されるか知りたい
    - ポインタ経由の呼び出し

      ret = foo->bar_func();

trace準備

- Debug Filesystem(debugfs)のmount
  - /etc/fstabに記述する場合

    debugfs /sys/kernel/debug debugfs defaults 0 0
  - mount -t debugfs nodev /sys/kernel/debug
  - 多くの場合はmount処理は自動で処理済み
- Debugfsでmountした /sys/kernel/debug/tracing以下のファイルがインタフェース

Tracerインタフェース(1)

```
~# cd /sys/kernel/debug/
/sys/kernel/debug# ls
asoc                gpio          pm_qos              suspend_stats
bdi                 hid           pvr                 tee
clk                 kprobes       pwm                 tracing
debug_enabled       memblock      ras                 usb
dma_buf             mmc0          regmap              wakeup_sources
dri                 mmc1          regulator           xf-dbg-trace
dynamic_debug       mmc2          sched_features
extfrag             opp           sleep_time
fault_around_bytes  pinctrl       split_huge_pages
```

Tracerインタフェース(2)

```
/sys/kernel/debug# cd tracing
/sys/kernel/debug/tracing# ls
README                      free_buffer         set_event          trace_marker
available_events            instances           set_event_pid      trace_options
available_filter_functions  kprobe_events       set_ftrace_filter  trace_pipe
available_tracers           kprobe_profile      set_ftrace_notrace tracing_cpumask
buffer_size_kb              max_graph_depth     set_ftrace_pid     tracing_max_latency
buffer_total_size_kb        options             set_graph_function tracing_on
current_tracer              per_cpu             set_graph_notrace  tracing_thresh
dyn_ftrace_total_info       printk_formats      snapshot           uprobe_events
enabled_functions           saved_cmdlines      trace              uprobe_profile
events                      saved_cmdlines_size trace_clock
```

tracing_on

- トレース機能のON/OFF

  # cat tracing_on

  1　　ON状態

  # echo 0 > tracing_on　OFFに設定

  # cat tracing_on

  0　　OFF状態

- 初期はONなのでまずOFFにしておく

available_tracers

- 利用できるTracerの表示

  # cat available_tracers

  function_graph preemptirqsoff preemptoff irqsoff function nop

current_tracer

- 利用するTracerの設定/表示

  # cat current_tracer

  nop　　デフォルトはnop

  # echo function_graph > current_tracer

  # cat current_tracer

  function_graph

  # echo function > current_tracer

  function

trace

- Trace結果出力

```
# cat trace
# tracer: function
#
# entries-in-buffer/entries-written: 0/0   #P:4
#
#                                _-----=> irqs-off
#                               / _----=> need-resched
#                              | / _---=> hardirq/softirq
#                              || / _--=> preempt-depth
#                              ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
```

  - まだOFFなのでヘッダーのみ

trace実行

- あまり良くない方法

  # echo 1 > tracing_on

  # run_test

  # echo 0 > tracing_on

- 良い方法

  # echo 1 > tracing_on; run_test; echo 0 > tracing_on

- ちょっとした時間で膨大なtraceが蓄積される
- リングバッファなので重要な結果が上書きされてしまう

trace実行例

- R-Car USB On-The-Go機能における Host->Peripheral切り替え
- 切替処理はshell scriptでsysfsを操作して実行
- traceは最後の切り替える瞬間のみ

  echo 1 > tracing_on;

  echo “e6590000.usb” > /sys/kernel/config/usb_gadget/g1/UDC;

  echo 0 > tracing_on (実際は1行)
- trace結果は44736エントリ

traceを見る方法

- traceが想定される関数をあらかじめ予想しておく
  - kernel内は機能毎に関数の先頭が統一されていることが多いので特徴的な部分だけでもよい
  - R-Car USBドライバではusbhs
- trace結果に予想関数が含まれているか確認
  - 含まれて無ければ
    - 予想が間違っている
    - 処理が長くバッファが上書きされた

```
rca-2030  [000] ...1  249.808851: usb_ep_autoconfig_reset <-configfs_composite_bind
rca-2030  [000] ...1  249.808853: usb_ep_autoconfig_reset <-configfs_composite_bind
rca-2030  [000] ...1  249.808855: usbhsg_gadget_start <-udc_bind_to_driver
sys-2065  [002] ...3  249.808856: flush_dcache_page <-__cpu_copy_user_page
sys-2065  [002] ...3  249.808856: preempt_count_sub <-wp_page_copy.isra.12
sys-2065  [002] ...2  249.808856: preempt_count_sub <-wp_page_copy.isra.12
sys-2065  [002] ...1  249.808857: mem_cgroup_try_charge <-wp_page_copy.isra.12
sys-2065  [002] ...1  249.808857: get_mem_cgroup_from_mm <-mem_cgroup_try_charge
rca-2030  [000] ...1  249.808858: rcar_gen3_phy_usb2_set_peripheral <-usbhsg_gadget_start
sys-2065  [002] ...1  249.808858: __rcu_read_lock <-get_mem_cgroup_from_mm
sys-2065  [002] ...1  249.808858: mem_cgroup_from_task <-get_mem_cgroup_from_mm
sys-2065  [002] ...1  249.808859: __rcu_read_unlock <-get_mem_cgroup_from_mm
sys-2065  [002] ...1  249.808859: try_charge <-mem_cgroup_try_charge
rca-2030  [000] ...1  249.808859: usbhs_irq_callback_update <-usbhsg_gadget_start
sys-2065  [002] ...1  249.808859: _raw_spin_lock <-wp_page_copy.isra.12
sys-2065  [002] ...1  249.808860: preempt_count_add <-_raw_spin_lock
rca-2030  [000] ...1  249.808860: usbhs_write <-usbhs_irq_callback_update
sys-2065  [002] ...2  249.808860: ptep_clear_flush <-wp_page_copy.isra.12
sys-2065  [002] ...2  249.808861: page_add_new_anon_rmap <-wp_page_copy.isra.12
sys-2065  [002] ...2  249.808861: __mod_zone_page_state <-page_add_new_anon_rmap
sys-2065  [002] ...2  249.808862: __page_set_anon_rmap <-page_add_new_anon_rmap
rca-2030  [000] ...1  249.808862: usbhs_mod_is_host <-usbhs_irq_callback_update
```

set_ftrace_filter

- 関数名フィルタ機能

```
# echo SyS* > set_ftrace_filter
...
# cat trace
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
...
        test.sh-2039  [000] ...1   461.423684: SyS_rt_sigprocmask <-el0_svc_naked
        test.sh-2039  [000] ...1   461.423697: SyS_lseek <-el0_svc_naked
        test.sh-2039  [000] ...1   461.423706: SyS_clone <-el0_svc_naked
        test.sh-2039  [000] ...1   461.424319: SyS_rt_sigprocmask <-el0_svc_naked
          <...>-2040  [001] ...1   461.424335: SyS_close <-el0_svc_naked
        test.sh-2039  [000] ...1   461.424396: SyS_rt_sigprocmask <-el0_svc_naked
```

- 追加して複数登録も可能

```
# echo schdule* >> set_ftrace_filter
```

set_ftrace_pid

- PIDフィルタ機能

  # echo 1001 > set_ftrace_pid

  # cat set_ftrace_pid

  1001

  - プロセスID 1001のtraceのみ保存
  - Shell scriptではecho $$ > set_ftrace_pidで自PIDを指定可能
  - 設定は1PIDのみ

buffer_size_kb

- バッファサイズ変更機能(単位kByte)

```
# cat buffer_size_kb
1408
# cat per_cpu/cpu0/buffer_size_kb
1408
# echo 10000 > buffer_size_kb
# cat buffer_size_kb
10000
# cat per_cpu/cpu0/buffer_size_kb
10000
```

参考

- Documentation/trace/ftrace.txt
- Documentation/trace/ftrace-design.txt
- kernel/trace/ftrace.c
- arch/arm64/include/asm/ftrace.h
- arch/arm64/kernel/entry-ftrace.S
- arch/arm64/kernel/ftrace.c
- include/linux/ftrace.h